PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :
G06F 3/06, G11B 20/18
G06F 11/10

A1

(11) International Publication Number: WO 93/13475

(43) International Publication Date: 8 July 1993 (08.07.93)

(21) International Application Number: PCT/US92/11283

(22) International Filing Date: 18 December 1992 (18.12.92)

(30) Priority data:
814,000 27 December 1991 (27.12.91) US

(71) Applicant: COMPAQ COMPUTER CORPORATION [US/US]; 20555 State Highway 249, Houston, TX 77070 (US).

(72) Inventor: NEUFELD, E., David ; 15618 Downford Drive, Tomball, TX 77375 (US).

(74) Agent: CABELLO, J., David; Compaq Computer Corporation, Mail Stop 060803, 20555 State Highway 249, Houston, Texas 77070 (US).

(81) Designated States: AT, AU, BG, BR, CA, CH, CS, DE, DK, ES, FI, GB, HU, JP, KR, NL, NO, PL, RO, RU, SE, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD FOR PERFORMING DISK ARRAY OPERATIONS USING A NONUNIFORM STRIPE SIZE MAPPING SCHEME

(57) Abstract

A method and apparatus for improving disk performance in a disk array subsystem. A nonuniform mapping scheme is used wherein the disk array includes regions having varying sizes of data stripes. The disk array includes a region comprised of data stripes having a stripe size that corresponds to the size of the internal data structures frequently used by the file system, in addition to a region comprised of a number of data stripes having a larger stripe size which are used for general data storage. When a write operation occurs involving one of the data structures, the data structure is preferably mapped to the small stripe region in the disk array having a size which matches the size of the data structure. In this manner, whenever a file system data structure is updated, the operation is a full stripe write. This removes the performance penalty associated with partial stripe write operations.

MODIFIED 3 + 1 MAPPING SCHEME
FOR 2 kB BLOCKS

	DISK 0	DISK 1	DISK 2	DISK 3
STRIPE 0	0	2	UNUSED	P0
	1	3	UNUSED	P1
STRIPE 1	4	6	UNUSED	P2
	5	7	UNUSED	P3
STRIPE 2	8	10	UNUSED	P4
	9	11	UNUSED	P5
STRIPE 3	12	16	20	P6
	13	17	21	P7
	14	18	22	P8
	15	19	23	P9
STRIPE 4	24	28	41	P10
	25	29	42	P11
	26	30	43	P12
	27	40	44	P13
STRIPE 5	45	49	53	P14
	46	50	54	P15
	47	51	55	P16
	48	52	56	P17
STRIPE 6	57	61	65	P18
	58	62	66	P19
	59	63	67	P20
	60	64	68	P21

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CS	Czechoslovakia
CZ	Czech Republic
DE	Germany
DK	Denmark
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakhstan
LI	Liechtenstein
LK	Sri Lanka
LU	Luxembourg
MC	Monaco
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SK	Slovak Republic
SN	Senegal
SU	Soviet Union
TD	Chad
TG	Togo
UA	Ukraine
US	United States of America
VN	Viet Nam

METHOD FOR PERFORMING DISK ARRAY OPERATIONS USING A NONUNIFORM STRIPE SIZE MAPPING SCHEME

The present invention is directed toward a method for improving performance for multiple disk drives in computer systems, and more particularly to a method for performing write operations in a disk array utilizing parity data redundancy and recovery protection.

Microprocessors and the computers which utilize them have become increasingly more powerful during the recent years. Currently available personal computers have capabilities in excess of the mainframe and minicomputers of ten years ago. Microprocessor data bus sizes of 32 bits are widely available whereas in the past 8 bits was conventional and 16 bits was common.

Personal computer systems have developed over the years and new uses are being discovered daily. The uses are varied and, as a result, have different requirements for various subsystems forming a complete computer system. With the increased performance of computer systems, it became apparent that mass storage subsystems, such as fixed disk drives, played an increasingly important role in the transfer of data to and from the computer system. In the past few years, a new trend in storage subsystems, referred to as a disk array subsystem, has emerged for improving data transfer performance, capacity and reliability. One reason for building a disk array subsystem is to create

a logical device that has a very high data transfer rate. This may be accomplished by "ganging" multiple standard disk drives together and transferring data to or from these drives in parallel. Accordingly, data is stored "across" each of the disks comprising the disk array so that each disk holds a portion of the data comprising a data file. If n drives are ganged together, then the effective data transfer rate may be increased up to n times. This technique, known as striping, originated in the supercomputing environment where the transfer of large amounts of data to and from secondary storage is a frequent requirement. In striping, a sequential data block is broken into segments of a unit length, such as sector size, and sequential segments are written to sequential disk drives, not to sequential locations on a single disk drive. The unit length or amount of data that is stored "across" each disk is referred to as the stripe size. The stripe size affects data transfer characteristics and access times and is generally chosen to optimize data transfers to and from the disk array. If the data block is longer than n unit lengths, the process repeats for the next stripe location on the respective disk drives. With this approach, the n physical drives become a single logical device and may be implemented either through software or hardware.

One technique that is used to provide for data protection and recovery in disk array subsystems is referred to as a parity scheme. In a parity scheme, data blocks being written to various drives within the array are used and a known EXCLUSIVE-OR (XOR) technique is used to create parity information which is written to a reserved or parity drive within the array. The advantage of a parity scheme is that it may be used to

minimize the amount of data storage dedicated to data redundancy and recovery purposes within the array. For example, Figure 1 illustrates a traditional 3+1 mapping scheme wherein three disks, disk 0, disk 1 and disk 2, are used for data storage, and one disk, disk 3, is used to store parity information. In Figure 1, each rectangle enclosing a number or the letter "p" coupled with a number corresponds to a sector, which is preferably 512 bytes. As shown in Figure 1, each complete stripe uses four sectors from each of disks 0, 1 and 2 for a total of 12 sectors of data storage per disk. Assuming a standard sector size of 512 bytes, the stripe size of each of these disk stripes, which is defined as the amount of storage allocated to a stripe on one of the disks comprising the stripe, is 2 kbytes. Thus each complete stripe, which includes the total of the portion of each of the disks allocated to a stripe, can store 6 kbytes of data. Disk 3 of each of the stripes is used to store parity information. However, there are a number of disadvantages to the use of parity fault tolerance techniques in disk array systems. One disadvantage to the 3+1 mapping scheme illustrated in Figure 1 is the loss of performance within the disk array as the parity drive must be updated each time a data drive is updated or written to. The data must undergo the XOR process in order to write the updated parity information to the parity drive as well as writing the data to the data drives. This process may be partially alleviated by having the parity data also distributed, relieving the load on the dedicated parity disk. However, this would not reduce the number of overall data write operations.

Another disadvantage to parity fault tolerance techniques is that traditional operating systems perform many small writes to the disk subsystem which

are often smaller than the stripe size of the disk array, referred to as partial stripe write operations. For example, many traditional file systems use small data structures to represent the structure of the files, directories, and free space within the file system. In a typical UNIX file system, this information is kept in a structure called an INODE, which is generally 2 kbytes in size. The INODE or INDEX NODE contains information on the type and size of the file, where the data is located, owning and using users, the last time the file was modified or accessed, the last time the NODE was modified, and the number of links or file names associated with that INODE. In the OS/2 high performance file system, this structure is called an FNODE. These structures are updated often since they contain file access and modification dates and file size information. These structures are relatively small compared with typical data stripe sizes used in disk arrays, thus resulting in a large number of partial stripe write operations.

When a large number of partial stripe write operations occur in a disk array, the performance of the disk subsystem is seriously impacted because, as explained below, the data or parity information currently on the disk must be read off of the disk in order to generate the new parity information. This results in extra revolutions of the disk drive and causes delays in servicing the request. In addition to the time required to perform the actual operations, it will be appreciated that a READ operation followed by a WRITE operation to the same sector on a disk results in the loss of one disk revolution, or approximately 16.5 milliseconds for certain types of hard disk drives.

Where a complete stripe of data is being written to the array, the parity information may be generated

directly from the data being written to the drive array, and therefore no extra read of the disk stripe is required. However, as mentioned above, a problem occurs when the computer writes only a partial stripe to the disk array because the disk array controller does not have sufficient information from the data to be written to compute parity for the complete stripe. Thus, partial stripe write operations generally require data stored on a disk to first be read, modified by the process active on the host system, and written back to the same address on the data disk. This operation consists of a data disk READ, modification of the data, and a data disk WRITE to the same address. There are generally two techniques used to compute parity information for partial stripe write operations.

In the first technique, a partial stripe write to a data disk in an XOR parity fault tolerant system includes issuing a READ command in order to maintain parity fault tolerance. The computer system first reads the parity information from the parity disk for the data disk sectors which are being updated and the old data values that are to be replaced from the data disk. The XOR parity information is then recalculated by the host or a local processor, or dedicated logic, by XORing the old data sectors to be replaced with the related parity sectors. This recovers the parity value without those data values. The new data values are XORed on to this recovered value to produce the new parity data. A WRITE command is then executed, writing the updated data to the data disks and the new parity information to the parity disk. It will be appreciated that this process requires two additional partial sector READ operations, one from the parity disk and one reading the old data, prior to the generation of the new XOR parity information. Additionally, the

WRITE operations are to locations which have just been read. Consequently, data transfer performance suffers.

The second method requires reading the remainder of the data that is not to be repudiated for the stripe, despite the fact that it is not being replaced by the WRITE operation. Using the new data and the old data which has been retrieved, the new parity information may be determined for the entire stripe which is being updated. This process requires a READ operation of the data not to be replaced and a full stripe WRITE operation to save the parity information.

According to the prior art, a disk array utilizing parity fault tolerance had to perform one of the above techniques to manage partial stripe WRITE operations. Therefore, partial stripe writes hurt system performance because either the remainder of the stripe that is not being written must be fetched or the existing parity information for the stripe must be read prior to the actual write of the information. Accordingly, there exists a need for an improved method for performing disk WRITE operations in a parity fault tolerant disk array in order to decrease the number of partial stripe write operations.

Background on disk drive formatting is deemed appropriate. When a disk drive is produced or manufactured, the manufacturer will also generally have low level formatted the disk. A low level format operation involves the creation of sectors on the disk along with their address markings, which are used to identify the sectors after the formatting is completed. The data portion of the sector is established and filled in with dummy data. When a disk drive unit is incorporated into a computer system, the disk controller and the respective operating system in control of the computer system must perform a high

level or logical format of the disk drive to place the "file system" on the disk and make the disk drive conform to the standards of the operating system. This high level formatting is performed by the respective disk controller in conjunction with an operating system service referred to as a "make file system" program. In the UNIX operating system, the make file system program works in conjunction with the disk controller to create the file system on the disk array. In traditional systems, the operating system views the disk as a sequential list of blocks or sectors, and the make file system program is unaware as to the topology of these blocks.

The present invention is directed toward a method and apparatus for improving disk performance in a computer system having a disk array subsystem. In the method according to the present invention, a nonuniform mapping scheme is used wherein the disk array includes certain designated regions having varying sizes of data stripes. The disk array includes a region comprised of a number of data stripes having a stripe size that is approximately the same as the size of internal data structures frequently used by the file system, in addition to a region which includes a number of data stripes having a larger stripe size which are used for general data storage. When a write operation occurs involving one of the small data structures, the data structure is preferably mapped to the small stripe region in the disk array wherein the complete stripe size matches the size of the data structure. In this manner, whenever the file system data structure is updated, the operation is a full stripe write. This reduces the number of partial stripe write operations,

thus reducing the performance penalty associated with these operations.

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

Figure 1 is a prior art diagram of a traditional 3+1 disk array mapping scheme having a uniform stripe size;

Figures 2 and 3 are block diagrams of an illustrative computer system on which the method of the present invention may be practiced;

Figure 4 is a block diagram of the disk subsystem of the preferred embodiment;

Figure 5 is a functional block diagram of the transfer controller of Fig. 4 according to the preferred embodiment;

Figure 6 is a diagram of a 3+1 disk array mapping scheme having varying stripe sizes according to a first embodiment;

Figure 7 is a diagram of a RAID 5 3+1 disk array mapping scheme having varying stripe sizes according to a second embodiment of the invention;

Figure 8 is a diagram of a 4+1 disk array mapping scheme according to the preferred embodiment of the invention;

Figure 9 is a flowchart diagram of a WRITE operation according to the method of the present invention; and

Figure 10 is a flowchart diagram of a READ operation according to the method of the present invention.

The computer system and disk array subsystem described below represent the preferred embodiment of the present invention. It is also contemplated that other computer systems, not having the capabilities of the system described below, may be used to practice the present invention.

Referring now to Figs. 2 and 3, the letter C generally designates a computer system on which the present invention may be practiced. For clarity, system C is shown in two portions with the interconnections between Figs. 2 and 3 designated by reference to the circled numbers 1 to 8. System C is comprised of a number of block elements interconnected via four buses.

A central processing unit CPU comprises a system processor 20, a numerical co-processor 22, a cache memory controller 24, and associated logic circuits connected to a system processor bus 26. Associated with cache controller 24 is a high speed cache data random access memory (RAM) 28, non-cacheable memory address (NCA) map programming logic circuitry 30, non-cacheable address or NCA memory map 32, address exchange latch circuitry 34, data exchange transceiver 36 and page hit detect logic 43. Associated with the CPU also are system processor ready logic circuit 38, next address (NA) enable logic circuit 40 and bus request logic circuit 42.

The system processor is preferably an Intel Corporation 80386 microprocessor. The system processor 20 has its control, address and data lines interfaced to the system processor bus 26. The co-processor 22 is preferably an Intel 80387 and/or Weitek WTL3167 numerical processor interfacing with the local processor bus 26 and the system processor 20 in the

conventional manner. The cache RAM 28 is preferably a suitable high-speed static random access memory which interfaces with the address and data elements of bus 26 under the control of the cache controller 24 to carry out required cache memory operations. The cache controller 24 is preferably an Intel 82385 cache controller configured to operate in two-way set associative master mode. In the preferred embodiment, the components are the 33 MHz versions of the respective units. An Intel 80486 microprocessor and an external cache memory system may replace the 80386, numeric coprocessor, 82385 and cache RAM if desired. Address latch circuitry 34 and data transceiver 36 interface the cache controller 34 with the processor 20 and provide a local bus interface between the processor bus 26 and a host or memory bus 44. Circuit 38 is a logic circuit which provides a bus ready signal to control access to the bus 26 and indicate when the next cycle may begin. The enable circuit 40 is utilized to indicate that the next address of data or code to be utilized by sub-system elements in pipelined address mode may be placed on the local bus 26.

Non-cacheable memory address (NCA) map programmer 30 cooperates with the processor 20 and the non-cacheable address memory 32 to map non-cacheable memory locations. The non-cacheable address memory 32 is utilized to designate areas of the system memory that are non-cacheable to avoid various types of cache coherency problems. The bus request logic circuit 42 is utilized by the processor 20 and associated elements to request access to the host bus 44 in situations such as when requested data is not located in cache memory 28 and access to system memory is required.

The main memory array or system memory 58 is coupled to the host bus 44. The main memory array 58

is preferably dynamic random access memory. Memory 58 interfaces with the host bus 44 via EISA bus buffer (EBB) data buffer circuit 60, a memory controller circuit 62, and a memory mapper 68. The buffer 60 performs data transceiving and parity generating and checking functions. The memory controller 62 and memory mapper 68 interface with the memory 58 via address multiplexor and column address strobe (ADDR/CAS) buffers 66 and row address strobe (RAS) enable logic circuit 64.

In the drawings, System C is configured as having the processor bus 26, the host bus 44, an extended industry standard architecture (EISA) bus 46 (Fig. 3) and an X bus 90 (Fig. 3). The details of the portions of the system illustrated in Fig. 3 and not discussed in detail below are not significant to the present invention other than to illustrate an example of a fully configured computer system. The portion of System C illustrated in Fig. 3 is essentially a configured EISA system which includes the necessary EISA bus 46 and EISA bus controller 48, data latches and transceivers referred to as EBB data buffers 50 and address latches and buffers 52 to interface between the EISA bus 46 and the host bus 44. Also illustrated in Fig. 2 is an integrated system peripheral (ISP) 54, which incorporates a number of the elements used in an EISA-based computer system.

The integrated ISP 54 includes a direct memory access controller 56 for controlling access to main memory 58 (Fig. 1) or memory contained in an EISA slot and input/output (I/O) locations without the need for access to the processor 20. The ISP 54 also includes interrupt controllers 70, non-maskable interrupt logic 72, and system timer 74 which allow control of interrupt signals and generate necessary timing signals

and wait states in a manner according to the EISA specification and conventional practice. In the preferred embodiment, processor generated interrupt requests are controlled via dual interrupt controller circuits emulating and extending conventional Intel 8259 interrupt controllers. The ISP 54 also includes bus arbitration logic 75 which, in cooperation with the bus controller 48, controls and arbitrates among the various requests for EISA bus 46 by the cache controller 24, the DMA controller 56, and bus master devices located on the EISA bus 46.

The EISA bus 46 includes ISA and EISA control buses 76 and 78, ISA and EISA data buses 80 and 82, and are interfaced via the X bus 90 in combination with the ISA control bus 76 from the EISA bus 46. Control and data/address transfer for the X bus 90 are facilitated by X bus control logic 92, data buffers 94 and address buffers 96.

Attached to the X bus are various peripheral devices such as keyboard/mouse controller 98 which interfaces with the X bus 90 with a suitable keyboard and a mouse via connectors 100 and 102, respectively. Also attached to the X bus are read only memory (ROM) circuits 106 which contain basic operation software for the system C and for system video operations. A serial port communications port 108 is also connected to the system C via the X bus 90. Floppy disk support, a parallel port, a second serial port, and video support circuits are provided in block circuit 110.

The computer system C includes a disk subsystem 111 which includes a disk array controller 112, fixed disk connector 114, and fixed disk array 116. The disk array controller 112 is connected to the EISA bus 46, preferably in a slot, to provide for the communication of data and address information through the EISA bus

46. Fixed disk connector 114 is connected to the disk array controller 112 and is in turn connected to the fixed disk array 116.

Referring now to Fig. 4, the disk subsystem 111 used to illustrate the method of the present invention is shown. The disk array controller 112 has a local processor 130, preferably an Intel 80186. The local processor 130 has a multiplexed address/data bus UAD and control outputs UC. The multiplexed address/data bus UAD is connected to a transceiver 132 whose output is the local processor data bus UD. The multiplexed address/data bus UAD is also connected to the D inputs of a latch 134 whose Q outputs form the local processor address bus UA. The local processor 130 has associated with it random access memory (RAM) 136 coupled via the multiplexed address/data bus UAD and the address data bus UA. The RAM 136 is connected to the processor control bus UC to develop proper timing signals. Similarly, read only memory (ROM) 138 is connected to the multiplexed address/data bus UAD, the processor address bus UA and the processor control bus UC. Thus, the local processor 130 has its own resident memory to control its operation and for its data storage. A programmable array logic (PAL) device 140 is connected to the local processor control bus UC to develop additional control signals utilized in the disk array controller 112.

The local processor address bus UA, the local processor data bus, UD and the local processor control bus UC are also connected to a bus master interface controller (BMIC) 142. The BMIC 142 serves the function of interfacing the disk array controller 112 with a standard bus, such as the EISA or MCA bus, and acts as a bus master. In the preferred embodiment, the BMIC 142 is interfaced with the EISA bus 46 and is the

Intel 82355. Thus, by this connection with the local processor busses UA, UD and UC, the BMIC 142 can interface with the local processor 130 to allow data and control information to be passed between the host system C and the local processor 130.

Additionally, the local processor data bus UD and local processor control bus UC are preferably connected to a transfer controller 144. The transfer controller 144 is generally a specialized multi-channel direct memory access (DMA) controller used to transfer data between the transfer buffer RAM 146 and various other devices present in the disk array controller 112. For example, the transfer controller 144 is connected to the BMIC 142 by the BMIC data lines BD and the BMIC control lines BC. Thus, over this interface, the transfer controller 144 can transfer data from the transfer buffer RAM 146 to the BMIC 142 if a READ operation is requested. If a WRITE operation is requested, data can be transferred from the BMIC 142 to the transfer buffer RAM 146. The transfer controller 144 can then pass this information from the transfer buffer RAM 146 to disk array 116. The transfer controller 144 is described in greater detail in U.S. Application No. 431,735, and in its European counterpart, European Patent Office Publication No. 0427119, published April 4, 1991, which is hereby incorporated by reference.

The transfer controller 144 includes a disk data bus DD and a disk address bus and control bus DAC. The disk address and control bus DAC is connected to two buffers 165 and 166 which are part of the fixed disk connector 114 and are used to send and receive control signals between the transfer controller 144 and the disk array 116. The disk data bus DD is connected to two data transceivers 148 and 150 which are part of the

fixed disk connector 114. The outputs of the transceiver 148 and the transfer buffer 146 are connected to two disk drive port connectors 152 and 154. In similar fashion, two connectors 160 and 162 are connected to the outputs of the transceiver 150 and the buffer 166. Two hard disks can be connected to each connector 152, 154, 160, and 162. Thus, up to 8 disk drives can be connected and coupled to the transfer controller 144. As discussed below, in the preferred embodiment five disk drives are coupled to the transfer controller 144, and a 4+1 mapping scheme is used.

In the illustrative disk array system 112, a compatibility port controller (CPC) 164 is also connected to the EISA bus 46. The CPC 164 is connected to the transfer controller 144 over the compatibility data lines CD and the compatibility control lines CC. The CPC 164 is provided so that the software which was written for previous computer systems which do not have a disk array controller 112 and its BMIC 142, which are addressed over an EISA specific space and allow very high throughputs, can operate without requiring a rewriting of the software. Thus, the CPC 164 emulates the various control ports previously utilized in interfacing with hard disks.

Referring now to Fig. 5, the transfer controller 144 is itself comprised of a series of separate circuitry blocks. The transfer controller 144 includes two main units referred to as the RAM controller 170 and the disk controller 172. The RAM controller 170 has an arbiter to control the various interface devices that have access to the transfer buffer RAM 146 and a multiplexor so that the data can be passed to and from the transfer buffer RAM 146. Likewise, the disk controller 172 includes an arbiter to determine which

of the various devices has access to the integrated disk interface 174 and includes multiplexing capability to allow data to be properly transferred back and forth through the integrated disk interface 174.

The transfer controller 144 preferably includes 7 DMA channels. One DMA channel 176 is assigned to cooperate with the BMIC 142. A second DMA channel 178 is designed to cooperate with the CPC 164. These two devices, the BMIC 142 and the bus compatibility port controller 164, are coupled only to the transfer buffer RAM 146 through their appropriate DMA channels 176 and 178 and the RAM controller 170. The BMIC 142 and the compatibility port controller 164 do not have direct access to the integrated disk interface 174 and the disk array 116. The local processor 130 (Fig. 3) is connected to the RAM controller 170 through a local processor DMA channel 180 and is connected to the disk controller 172 through a local processor disk channel 182. Thus, the local processor 130 is connected to both the transfer buffer RAM 146 and the disk array 116 as desired.

Additionally, the transfer controller 144 includes 4 DMA disk channels 184, 186, 188 and 190 which allow information to be independently and simultaneously passed between the disk array A and the RAM 146. It is noted that the fourth DMA/disk channel 190 also includes XOR capability so that parity operations can be readily performed in the transfer controller 144 without requiring computations by the local processor 130. The above computer system C and disk array subsystem 111 represent the preferred computer system for the practice of the method of the present invention.

The computer system C preferably utilizes the UNIX operating system, although other operating systems may

be used. As described in the background, the UNIX operating system includes a service referred to as the make file system program. In the preferred embodiment, the make file system program provides information to the disk controller 112 as to how many INODEs are being created and the size of the INODEs. Optionally, the make file system includes sufficient intelligence to inform the disk controller 112 as to the desired stripe size in the small stripe and large stripe regions and the boundary separating these regions. As previously discussed, the number of INODEs is approximately equal to the number of files which are to be allowed in the system. The disk array controller 112 uses this information to develop the file system on each of the disks comprising the array 116.

The disk array controller 112 uses a multiple mapping scheme according to the present invention which partitions the disk array 116 into small stripe and large stripe regions. The small stripe region preferably occupies the first N sectors of each disk and is reserved for the INODE data structures, and the remaining stripes in the array form the large stripe region, which comprises free space used for data storage. Therefore, in the preferred embodiment, the disk controller 112 allocates the first N sectors of each of the disks in the array for the small stripe region. The remaining sectors of each of the disks are formatted into the large stripe region. The disk array controller 112 stores the boundary separating the small stripe and large stripe regions in the RAM 136. Thereafter, when the INODE data structures are written to each of the disks, the disk array controller 112 utilizes this boundary and writes the INODEs to the small stripe portion of the array 116. In this manner, whenever an INODE is updated, the resulting operation

is a full stripe write. This increases system performance because, as previously discussed, partial stripe write operations reduce performance because they generally require a preceding read operation. In an alternate embodiment of the invention, the small stripe region does not occupy the first N sectors of each disk, but rather the small stripe region includes a plurality of regions interspersed among the large stripe regions. In this embodiment, a plurality of boundaries which separate the small stripe and large stripe regions are stored in the RAM 136 so that the disk controller 112 can write the INODE data structures to the small stripe region.

In an alternate embodiment of the invention, the OS/2 operating system is used. In this embodiment, an OS/2 service similar to the make file system program discussed above provides information to the disk controller 112 as to how many FNODEs are being created and the size of the FNODEs. The disk controller then uses a multiple mapping scheme similar to that discussed above to partition the disk array 116 into small stripe and large stripe regions wherein the small stripe region is reserved for the FNODE data structures. It is noted that the present invention can operate in conjunction with any type of operating system or file system.

Referring again to Figure 1, when an INODE data structure having a size of 2 kbytes is written to a disk array having a uniform disk stripe size of 2 kbytes according to the prior art, for example stripe 0, then the entire INODE would be written to disk 0 in sectors 0, 1, 2 and 3, and disks 1 and 2 would be unused. Effectively, this operation negates the advantages of a disk array system since only one disk is being accessed. Also, the amount of unused space is

considerable, resulting in an inefficient use of the disk drives. If data is subsequently allowed to be written to the remainder of the stripe, i.e., the portion of the stripe residing in disks 1 and 2, then the resulting operation in this stripe will consist of a partial stripe write operation, which has performance penalties as described above.

Referring now to Figure 6, a diagram illustrating a 3+1 mapping scheme utilizing multiple stripe sizes according to one embodiment of the present invention is shown. Figure 6 is exemplary only, it being noted that the disk array 116 will utilize a much larger number of stripes of each size. The disk drives used in the preferred embodiment include a number of sectors each having 512 bytes of storage. In the embodiment shown in Figure 6, the disk array utilizes two stripe sizes, a disk stripe size of one kbyte and a disk stripe size of two kbytes. As shown in Figure 6, stripes 0, 1 and 2 utilize a disk stripe size of one kbyte using two sectors per disk in disks 0, 1 and 2 for a total of six sectors or 3 kbytes of data storage per complete stripe. In addition, stripes 0, 1, and 2 utilize two sectors in disk 3 to store parity information for each stripe. Stripes 3-6 utilize a 2 kbyte disk stripe size wherein four sectors per disk on disks 0, 1 and 2 are allocated for data storage for each stripe and four sectors on disk 3 are reserved for parity information for each stripe. In this embodiment, the INODE data structures are written to the portion of the disk array having the small stripe size, i.e., stripes 0, 1, or 2. As previously discussed, INODE structures are assumed to be 2 kbytes in size in the preferred embodiment.

Therefore, as shown in Figure 6, INODE structures written to stripes 0, 1 or 2 would fill up the area in disks 0 and 1, disk 2 would generally be unused, and

disk 3 would be used to store the respective parity information. In this embodiment, data would not be allowed to be written to disk 2 of the respective stripe after an INODE is written there, and thus partial stripe write operations are prevented from occurring. Therefore, by using a smaller stripe size in a portion of the disk array 116 and preventing data from being written to the unused space after an INODE is written, a write operation of these structures emulates a full stripe write. However, it is noted that disk 2 is unused or unwritten during this full stripe write, and thus an inefficient use of the disk area results. In addition, since disk 2 will generally be unused for each small stripe where an INODE structure is written, the data transfer bandwidth from the disk array system is reduced, and the array essentially operates as a 2+1 mapping scheme in these instances.

One solution to this problem is to distribute the unused space across different disks for each stripe as shown in Figure 7. In this manner, the reduction of data transfer bandwidth is not as significant since each disk is used approximately equally. However, data transfer bandwidth is sub-optimum since each disk access involves an unused disk. Furthermore, this method produces an undesirable amount of unused disk space.

In the preferred embodiment of the invention, a 4+1 mapping scheme is used, as shown in Figure 8. It is again noted that Figure 8 is exemplary only, and the disk array 116 of the preferred embodiment will utilize a much larger number of stripes in each of the small stripe and large stripe regions. The disk stripe size of the stripes in the small stripe region, stripes 0-4, wherein stripe size is defined as the amount of each

disk that is allocated to the stripe, is 512 bytes. In this manner, each complete stripe in the small stripe region holds exactly 2 kbytes of data, which is approximately equivalent to the size of an INODE structure. Accordingly, when an INODE structure is written to a small stripe in the disk array 116, a full stripe write operation is performed, and the INODE occupies the entire stripe without any unused space. In this manner, the bandwidth of the disk array 116 is optimally used because every disk participates in each access and no unused space results.

Referring again to Figure 4, in the preferred embodiment, a disk request is preferably submitted by the system processor 20 to the disk array controller 112 through the EISA bus 46 and BMIC 142. The local processor 130, on receiving this request through the BMIC 142, builds a data structure in the local processor RAM memory 136. This data structure is known as a command list and may be a simple READ or WRITE request directed to the disk array 116, or it may be a more elaborate set of requests containing multiple READ/WRITE or diagnostic and configuration requests. The command list is then submitted to the local processor 130 for processing. The local processor 130 then oversees the execution of the command list, including the transferring of data. Once the execution of the command list is completed, the local processor 130 notifies the operating system device driver running on the system microprocessor 20. The submission of the command list and the notification of the command list completion are achieved by a protocol which uses input/output (I/O) registers located in the BMIC 142.

The READ and WRITE operations executed by the disk array controller 112 are implemented as a number of application tasks running on the local processor 130.

Because of the nature of the interactive input/output operations, it is impractical for the illustrative computer system C to process disk commands as single batch tasks on the local processor 130. Accordingly, the local processor 130 utilizes a real time multi-tasking use system which permits multiple tasks to be addressed by the local processor 130, including the method of the present invention. Preferably, the operating system on the local processor 130 is the AMX86 multi-tasking executive by Kadak Products, Ltd. The AMX operating system kernel provides a number of system services in addition to the applications set forth in the method of the present invention.

Referring now to Figure 9, a flowchart diagram of a WRITE operation as carried out on a computer system C including the intelligent disk array controller 112 is shown. The WRITE operation begins at step 200, in which the active process or application causes the system processor 20 to generate a WRITE request which is passed to the disk device driver. The disk device driver is a portion of the software contained within the computer system C, preferably the system memory 58, which performs the actual interface operations with the disk units. The disk device driver software assumes control of the system processor 20 to perform specific tasks to carry out the required I/O operations. Control transfers to step 202, wherein the disk device driver assumes control of the system processor 20 and generates a WRITE command list.

In step 204, the device driver submits the WRITE command list to the disk controller 112 via the BMIC 142 or the CPC 164. The device driver then goes into a wait state to await a completion signal from the disk array controller 112. Logical flow of the operations proceeds to step 206, wherein the local processor 130

receives the WRITE command list and determines whether an INODE data structure is being written to the disk array 116. In making this determination, the local processor preferably utilizes the boundary between the small stripe and large stripe regions. In an alternate embodiment of the invention, intelligence is incorporated into the device driver wherein the device driver utilizes the boundary between the small stripe and large stripe regions and incorporates this information into the WRITE command list. If an INODE data structure is being written to the disk array 116, then in step 208 the local processor 130 builds disk specific WRITE instructions for the full stripe WRITE operation to the small stripe region. Control then transfers to step 210, wherein the transfer controller chip (TCC) 144 generates parity data from the INODE being written to the disk array 116. It is noted that the operation of writing the INODE to the small stripe region will be treated as a full stripe write operation, and thus no preceding READ operations associated with partial stripe write operations are encountered. Control of the operations then transfers to step 212, wherein the TCC 144 writes the data and the newly generated parity information to disks within the disk array 116. Control thereafter transfers to step 214, wherein the local processor 130 determines whether additional data is to be written to the disk array 116. If additional data is to be written to the disk array 116, control transfers to step 216 wherein the local processor 130 increments the memory addresses and decrements the number of bytes to be transferred. Control then returns to step 206. If no additional data is to be written to the disk array 116, control transfers from step 214 to step 224 where the local processor 130 signals WRITE complete.

If the local processor 130 receives the WRITE command list and determines that an INODE structure is not being written to the disk array 116, then in step 218 the local processor 130 builds disk specific WRITE instructions for the data to be written to the large stripe region. It is noted that this operation requires the local processor 130 to utilize the boundary between the small stripe and large stripe regions stored in the RAM 136 to develop the proper bias or offset to correct for the differing size stripes so that the proper physical disk addresses are developed. Optionally, this intelligence can be built into the device driver wherein the device driver has access to and utilizes the boundary between the small stripe and large stripe regions and incorporates this offset information into the WRITE command list. In this embodiment, the local processor 130 is not required to utilize the boundary between the small stripe and large stripe regions because this intelligence is incorporated into the device driver.

In step 220, the transfer controller chip 144 generates parity information solely for the data being written. Here it is noted that if the write operation will be a full stripe write, then the disk controller 112 can generate the parity information solely from the data to be written. However, if the write operation will be a partial stripe write operation, then a preceding read operation may need to be performed to read the data or parity information currently on the disk. As previously discussed, these additional read operations resulting from partial stripe write operations reduce the performance of the disk system 111. For techniques used to enhance the performance of partial stripe write operations, please see U.S. patent application serial number 752,773 titled METHOD FOR

PERFORMING WRITE OPERATIONS IN A PARITY FAULT TOLERANT DISK ARRAY" filed on August 30, 1991 and U.S. patent application serial number 815,118 titled "METHOD FOR IMPROVING PARTIAL STRIPE WRITE PERFORMANCE IN DISK ARRAY SUBSYSTEMS," filed on December 27, 1991, both of which are assigned to the same assignee as this invention and are hereby incorporated by reference. In step 222, the disk controller 112 writes the data and parity information to the large stripe region. Control then transfers to step 214 where the local processor 130 determines whether additional data is to be written to the disk array 116. If in step 214 it is determined that no additional data is to be transferred, control transfers to step 224, wherein the disk array controller 112 signals WRITE complete to the disk device driver. Control then passes to step 226, wherein the device driver releases control of the system processor 20 to continue execution of the application program. This completes operation of the WRITE sequence.

Referring now to Fig. 10, a READ operation as carried out on the disk array subsystem 111 using the intelligent disk array controller 112 is shown. The READ operation begins at step 250 when the active process or application program causes the system processor 20 to generate a READ command which is passed to the disk device driver. Control transfers to step 252, wherein the disk device driver assumes control of the system processor 20 and causes the system processor 20 to generate a READ command list similar to that described in U.S. patent application serial no. 431,737 assigned to Compaq Computer Corporation, assignee of the present invention. The READ command list is sent to the disk subsystem 111 in step 254, after which

operation the device driver waits until it receives a READ complete signal.

In step 256, the disk controller 112 receives the READ command list, via the BMIC 142 or CPC 146 and transfer controller 144, and determines if the read operation is intended to access data in the small stripe region, i.e., an INODE, or data in the large stripe region. In making this determination, the disk controller 112 preferably compares the disk address of the requested data with the boundary between the small stripe and large stripe regions stored in the RAM 136 to determine which region is being accessed. Optionally, more intelligence can be built into the device driver such that the device driver incorporates information as to which region is being accessed in the READ command list. According to this embodiment, the disk controller 112 would require little extra intelligence and would merely utilize this information in the READ command list in generating the disk specific READ requests.

If the small stripe region is being accessed, the local processor 130 generates disk specific READ requests for the requested INODE and its associated parity information in the small stripe region in step 260 and queues the requests in local RAM 136. Control transfers to step 264, wherein the requests are executed and the requested data is transferred from the disk array 112 through the transfer controller 144 and the BMIC 142 or the CPC 164 to the system memory 58 addresses indicated by the requesting task. If the disk controller 112 determines that the read operation is intended to access data in the large stripe region, the local processor 130 generates disk specific READ requests for the requested data and its associated parity information in the large stripe region in step

262 and queues the requests in local RAM 136. These requests are executed and the data transferred in step 264. Upon completion of the data transfer in step 264, the disk array controller 112 signals READ complete to the disk device driver in step 266, which releases control of the system processor 20.

Therefore, by providing varying stripe sizes in a disk array, and in particular providing a region with a complete stripe size that is equivalent to the size of small data structures that are often written to the disk array, the number of partial stripe write operations are reduced. By placing these small data structures into the small stripe region where the data stripes exactly match the size of the data structure, the resulting operation is a full stripe write. This increases disk performance because the performance penalties associated with partial stripe write operations are removed.

The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the components, methods and operation as well as in the details of the illustrated logic and flowcharts may be made without departing from the spirit of the invention.

Claims:

1. A method for improving disk array performance in a computer system disk array including a first region comprised of a plurality of data stripes having a first stripe size and a second region comprised of a plurality of data stripes having a stripe size larger than said first stripe size, wherein said first stripe size corresponds to the size of data structures used in the disk array, the method comprising:

generating a data write operation to the disk array;

determining if said data comprises one of the data structures;

writing said data to a data stripe in the first region having said first stripe size if said data comprises one of the data structures; and

writing said data to a data stripe in the second region if said data does not comprise one of the data structures.

2. The method of claim 1, wherein said data structure writing to the first region is a full stripe write operation.

3. The method of claim 1, wherein said step of data write determining is performed by a disk controller coupled to the disk array.

4. The method of claim 1, wherein said step of data write determining is performed by a system processor in the computer system.

5. The method of claim 1, further comprising:

generating a read operation to request data from the disk array;

determining whether said read operation is intended to access said first stripe size region or said second stripe size region; and

providing said requested data from either said first or second stripe size regions depending on said step of read operation determining.

6. The method of claim 1, wherein the disk array utilizes parity fault tolerance techniques, the method further comprising:

generating parity information for said data;

writing said parity information to the first region if said data comprises one of the data structures; and

writing said parity information to the second region if said data does not comprise one of the data structures.

7. A computer system which performs disk array write operations, comprising:

a system bus;

a disk array coupled to said system bus including a first region comprised of a plurality of data stripes having a first stripe size and a second region comprised of a plurality of data stripes having a stripe size larger than said first stripe size, wherein said first stripe size corresponds to the size of data structures used in the disk array;

means coupled to said system bus for generating a data write operation to the disk array;

means coupled to said system bus and said generating means for determining if said data comprises one of the data structures;

means coupled to said determining means and said system bus for writing said data to a data stripe

in said first stripe size region if said data comprises one of the data structures; and

means coupled to said determining means and said system bus for writing said data to a data stripe in said second stripe size region if said data does not comprise one of the data structures.

8. The apparatus of claim 7, wherein said data structure write operation to said first stripe size region is a full stripe write operation.

9. A method for improving disk array performance in a computer system disk array having a plurality of data stripes of varying sizes for storing data, wherein a first stripe size corresponds to the size of data structures used in the disk array, the method comprising:

generating a data write operation to the disk array;

determining if said data comprises one of the data structures;

writing said data to a data stripe having the first size corresponding to the size of the data structure if said data comprises one of the data structures; and

writing said data to a data stripe not having the first size if said data does not comprise one of the data structures.

10. The method of claim 9, wherein said data structure writing to said data stripe having the first size is a full stripe write operation.

11. The method of claim 9, wherein the disk array includes a region having the first stripe size for

storing the data structures and a region having a second stripe size, the method further comprising:

generating a read operation to request data from the disk array;

determining whether said read operation is intended to access said first stripe size region or said second stripe size region; and

providing said requested data from either said first or second stripe region depending on said read operation determining.

12. A computer system which performs disk array write operations, comprising:

a system bus;

a disk array coupled to said system bus having a plurality of data stripes of varying sizes for storing data, wherein a first stripe size corresponds to the size of data structures used in the disk array;

means coupled to said system bus for generating a data write operation to the disk array;

means coupled to said generating means and said system bus for determining if said data comprises one of the data structures;

means coupled to said determining means and said system bus for writing said data structure to a data stripe having said first size corresponding to the size of said data structure if said data comprises one of the data structures; and

means coupled to said determining means and said system bus for writing said data to a data stripe not having said first size if said data does not comprise one of the data structures.

13. A method for improving disk array performance in a computer system disk array which utilizes parity fault tolerance techniques, comprising:

creating a file system on the disk array,

partitioning the disk array to create a first region comprised of a plurality of data stripes having a first stripe size and a second region comprised of a plurality of data stripes having a stripe size larger than said first stripe size, wherein said first stripe size corresponds to the size of data structures used in the disk array;

generating a data write operation to the disk array;

determining if said data comprises one of the data structures;

writing said data to a data stripe in the first region having said first stripe size if said data comprises one of the data structures; and

writing said data to a data stripe in the second region if said data does not comprise one of the data structures.

14. The method of claim 13, wherein said data structure writing to the first region is a full stripe write operation.

15. The method of claim 13, further comprising:

generating a read operation to request data from the disk array;

determining whether said read operation is intended to access said first stripe size region or said second stripe size region; and

providing said requested data from either said first or second stripe size regions depending on said step of read operation determining.

A method and apparatus for improving disk performance in a disk array subsystem. A nonuniform mapping scheme is used wherein the disk array includes regions having varying sizes of data stripes. The disk array includes a region comprised of data stripes having a stripe size that corresponds to the size of the internal data structures frequently used by the file system, in addition to a region comprised of a number of data stripes having a larger stripe size which are used for general data storage. When a write operation occurs involving one of the data structures, the data structure is preferably mapped to the small stripe region in the disk array having a size which matches the size of the data structure. In this manner, whenever a file system data structure is updated, the operation is a full stripe write. This removes the performance penalty associated with partial stripe write operations.

TRADITIONAL 3 + 1 MAPPING SCHEME

	DISK 0	DISK 1	DISK 2	DISK 3
STRIPE 0	0	4	8	P0
	1	5	9	P1
	2	6	10	P2
	3	7	11	P3
STRIPE 1	12	16	20	P4
	13	17	21	P5
	14	18	22	P6
	15	19	23	P7
STRIPE 2	24	28	41	P8
	25	29	42	P9
	26	30	43	P10
	27	40	44	P11
STRIPE 3	45	49	53	P12
	46	50	54	P13
	47	51	55	P14
	48	52	56	P15
STRIPE 4	57	61	65	P16
	58	62	66	P17
	59	63	67	P18
	60	64	68	P19

PRIOR ART

FIG. 1

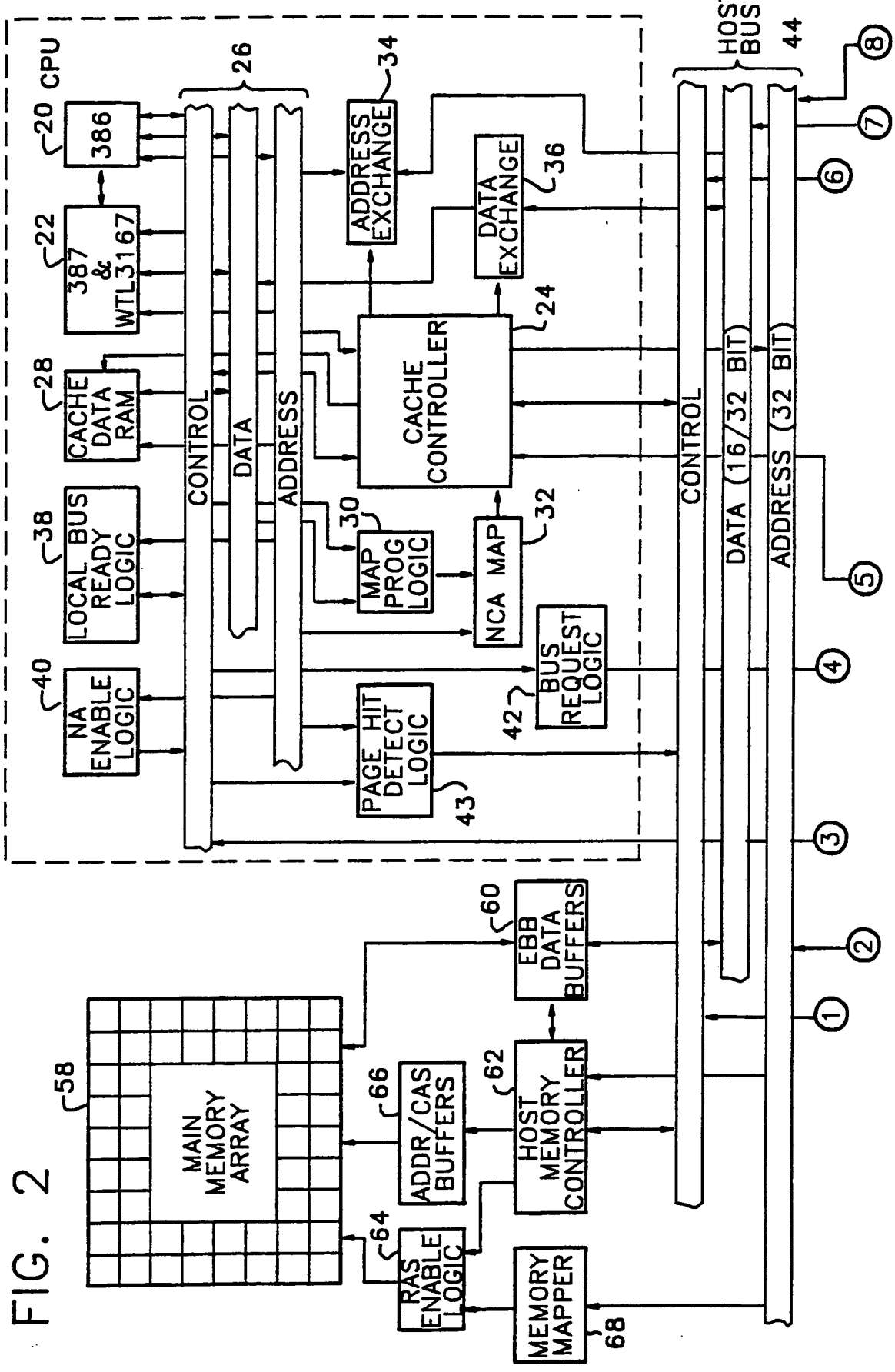
CPU
20
386
22
387 & WTL3167
28
CACHE DATA RAM
38
LOCAL BUS READY LOGIC
40
NA ENABLE LOGIC
26
CONTROL
DATA
ADDRESS
34
ADDRESS EXCHANGE
36
DATA EXCHANGE
24
CACHE CONTROLLER
30
MAP PROG LOGIC
32
NCA MAP
42
BUS REQUEST LOGIC
43
PAGE HIT DETECT LOGIC
HOST BUS
44
CONTROL
DATA (16/32 BIT)
ADDRESS (32 BIT)
8
7
6
5
4
3
2
1
60
EBB DATA BUFFERS
58
MAIN MEMORY ARRAY
66
ADDR/CAS BUFFERS
62
HOST MEMORY CONTROLLER
64
RAS ENABLE LOGIC
MEMORY MAPPER
68

FIG. 2

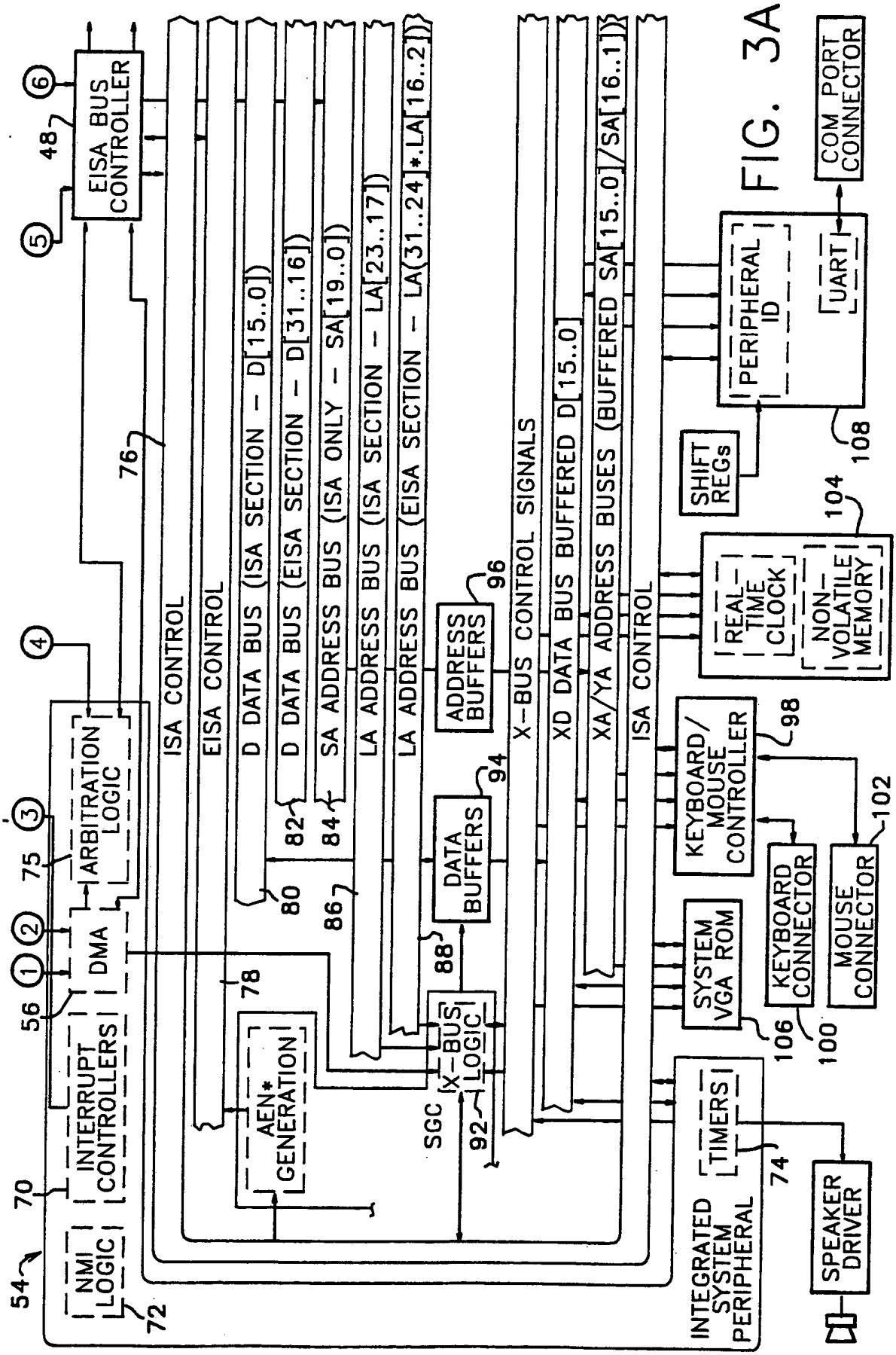
EISA BUS CONTROLLER
48
6
5
4
3
2
1
ARBITRATION LOGIC
INTERRUPT CONTROLLERS
DMA
NMI LOGIC
AEN* GENERATION
X-BUS LOGIC
SGC
TIMERS
INTEGRATED SYSTEM PERIPHERAL
SPEAKER DRIVER
ISA CONTROL
EISA CONTROL
D DATA BUS (ISA SECTION - D[15..0])
D DATA BUS (EISA SECTION - D[31..16])
SA ADDRESS BUS (ISA ONLY - SA[19..0])
LA ADDRESS BUS (ISA SECTION - LA[23..17])
LA ADDRESS BUS (EISA SECTION - LA[31..24]*.LA[16..2])
ADDRESS BUFFERS
DATA BUFFERS
X-BUS CONTROL SIGNALS
XD DATA BUS BUFFERED D[15..0]
XA/YA ADDRESS BUSES (BUFFERED SA[15..0]/SA[16..1])
ISA CONTROL
KEYBOARD/ MOUSE CONTROLLER
SYSTEM VGA ROM
KEYBOARD CONNECTOR
MOUSE CONNECTOR
SHIFT REGs
PERIPHERAL ID
UART
COM PORT CONNECTOR
REAL-TIME CLOCK
NON-VOLATILE MEMORY
54
70
72
75
76
78
80
82
84
86
88
92
94
96
98
100
102
104
106
108

FIG. 3A

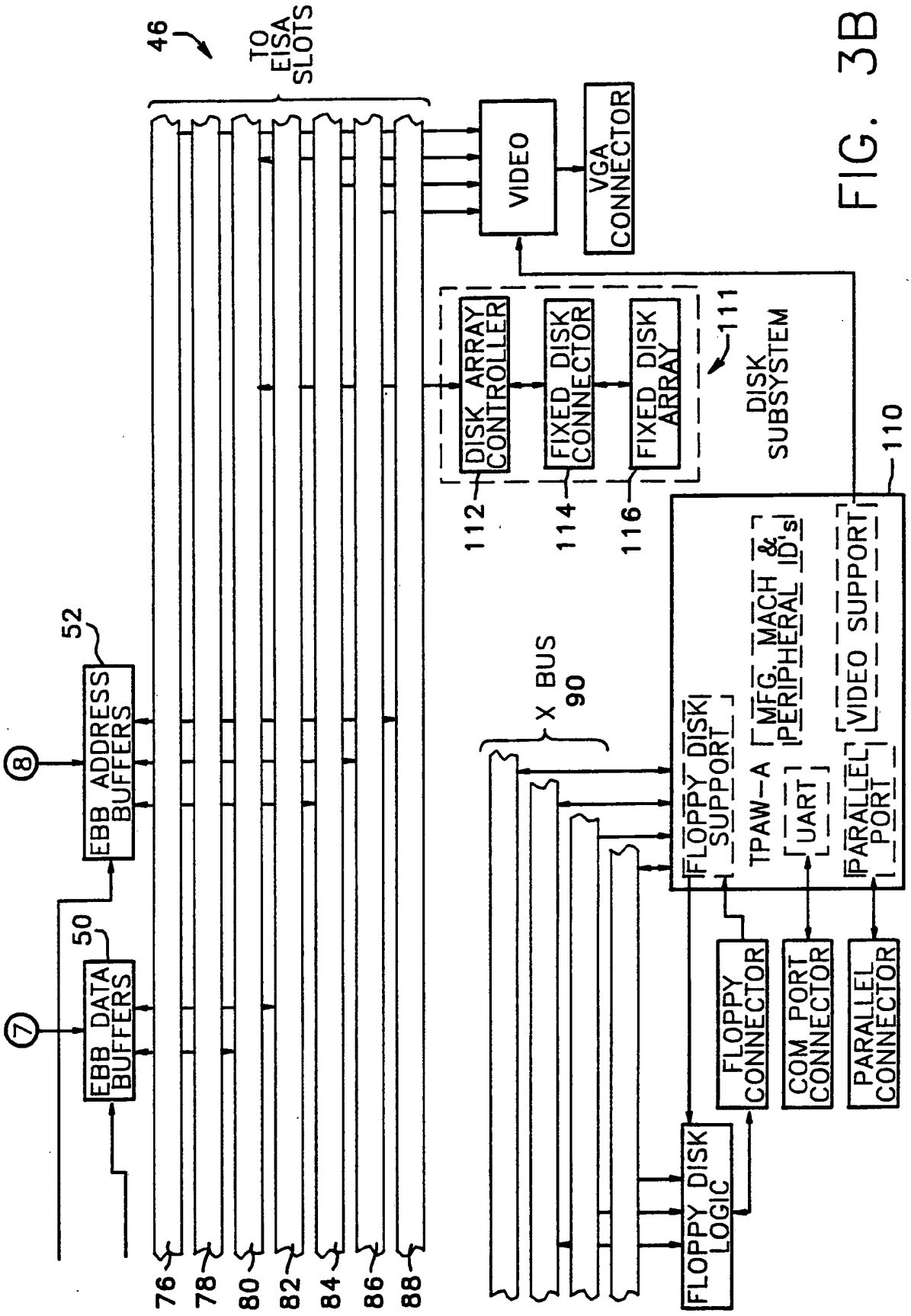
46
TO EISA SLOTS
VIDEO
VGA CONNECTOR
DISK ARRAY CONTROLLER
FIXED DISK CONNECTOR
FIXED DISK ARRAY
111
DISK SUBSYSTEM
110
112
114
116
52
EBB ADDRESS BUFFERS
50
EBB DATA BUFFERS
8
7
X BUS
90
FLOPPY DISK SUPPORT
TPAW-A
MFG. MACH & PERIPHERAL ID's
UART
PARALLEL PORT
VIDEO SUPPORT
FLOPPY CONNECTOR
COM PORT CONNECTOR
PARALLEL CONNECTOR
FLOPPY DISK LOGIC
76
78
80
82
84
86
88

FIG. 3B

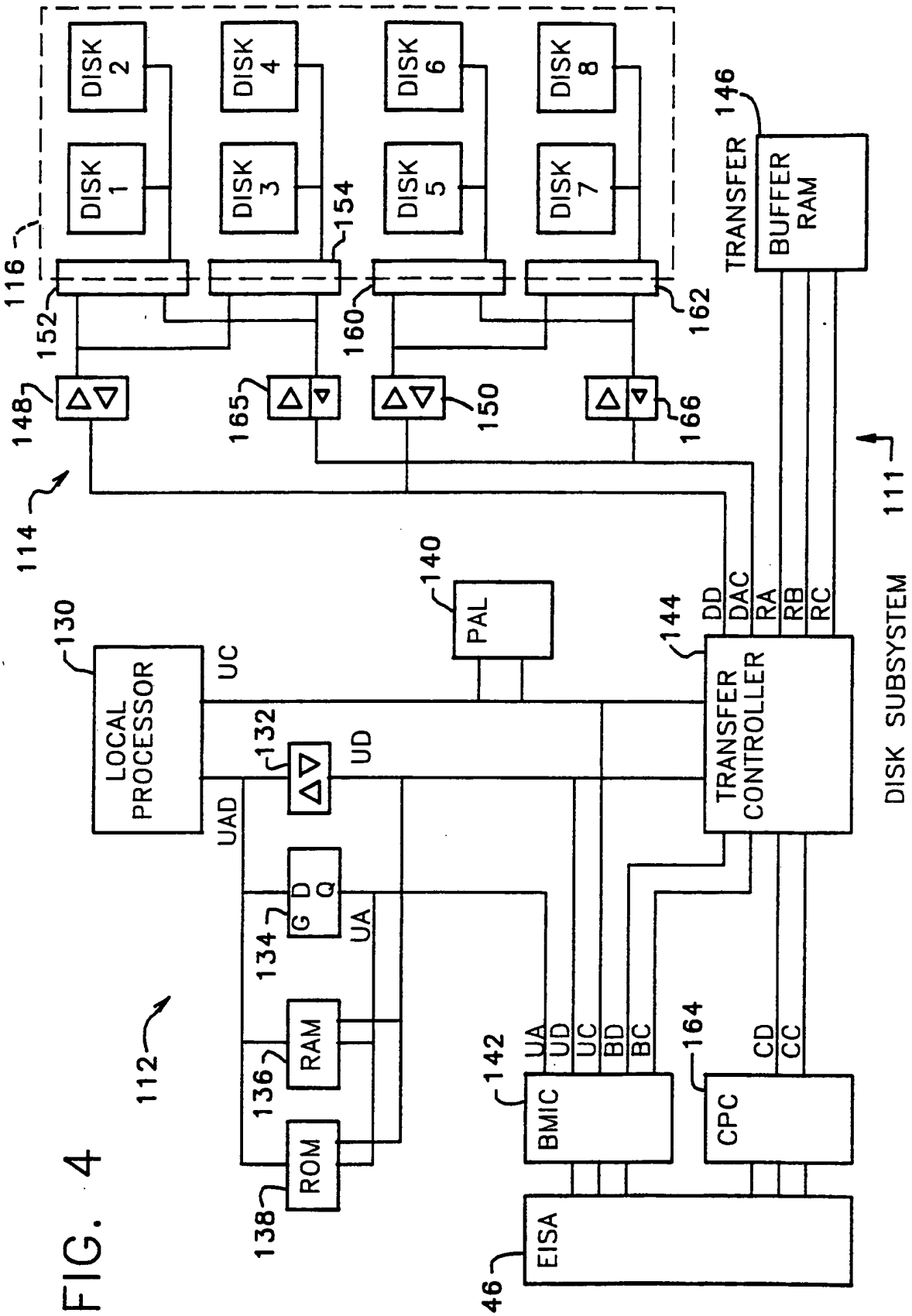
DISK 1
DISK 2
DISK 3
DISK 4
DISK 5
DISK 6
DISK 7
DISK 8
116
152
154
160
162
148
165
150
166
TRANSFER
146
BUFFER RAM
114
111
DISK SUBSYSTEM
140
PAL
130
LOCAL PROCESSOR
UC
UAD
132
UD
144
DD
DAC
RA
RB
RC
TRANSFER CONTROLLER
G
D
Q
134
UA
112
136
RAM
ROM
138
142
UA
UD
UC
BD
BC
BMIC
164
CD
CC
CPC
EISA
46

FIG. 4

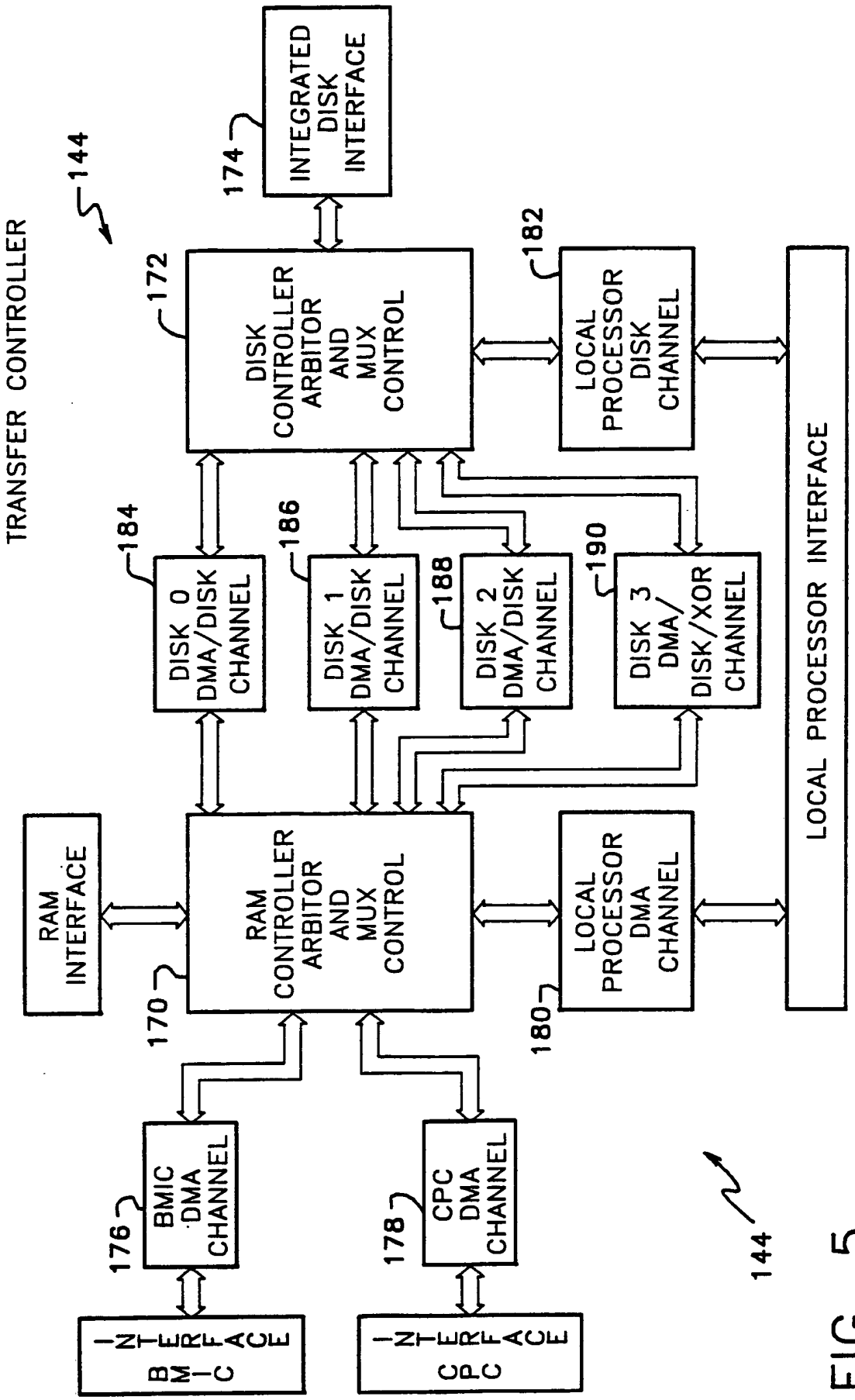
TRANSFER CONTROLLER
144
INTEGRATED DISK INTERFACE
174
DISK CONTROLLER ARBITOR AND MUX CONTROL
172
LOCAL PROCESSOR DISK CHANNEL
182
LOCAL PROCESSOR INTERFACE
DISK 0 DMA/DISK CHANNEL
184
DISK 1 DMA/DISK CHANNEL
186
DISK 2 DMA/DISK CHANNEL
188
DISK 3 DMA/ DISK/XOR CHANNEL
190
RAM INTERFACE
RAM CONTROLLER ARBITOR AND MUX CONTROL
170
LOCAL PROCESSOR DMA CHANNEL
180
BMIC DMA CHANNEL
176
CPC DMA CHANNEL
178
INTERFACE BM-C
INTERFACE CPC
144

FIG. 5

MODIFIED 3 + 1 MAPPING SCHEME FOR 2 kB BLOCKS

	DISK 0	DISK 1	DISK 2	DISK 3
STRIPE 0	0	2	UNUSED	P0
	1	3	UNUSED	P1
STRIPE 1	4	6	UNUSED	P2
	5	7	UNUSED	P3
STRIPE 2	8	10	UNUSED	P4
	9	11	UNUSED	P5
STRIPE 3	12	16	20	P6
	13	17	21	P7
	14	18	22	P8
	15	19	23	P9
STRIPE 4	24	28	41	P10
	25	29	42	P11
	26	30	43	P12
	27	40	44	P13
STRIPE 5	45	49	53	P14
	46	50	54	P15
	47	51	55	P16
	48	52	56	P17
STRIPE 6	57	61	65	P18
	58	62	66	P19
	59	63	67	P20
	60	64	68	P21

FIG. 6

RAID 5
MODIFIED 3 + 1 MAPPING SCHEME
FOR 2 kB BLOCKS

	DISK 0	DISK 1	DISK 2	DISK 3
STRIPE 0	0	2	UNUSED	P0
	1	3	UNUSED	P1
STRIPE 1	P2	4	6	UNUSED
	P3	5	7	UNUSED
STRIPE 2	UNUSED	P4	8	10
	UNUSED	P5	9	11
STRIPE 3	12	16	20	P6
	13	17	21	P7
	14	18	22	P8
	15	19	23	P9
STRIPE 4	P10	24	28	41
	P11	25	29	42
	P12	26	30	43
	P13	27	40	44
STRIPE 5	45	P14	49	53
	46	P15	50	54
	47	P16	51	55
	48	P17	52	56
STRIPE 6	57	61	P18	65
	58	62	P19	66
	59	63	P20	67
	60	64	P21	68

FIG. 7

MODIFIED 4 + 1 MAPPING SCHEME

	DISK 0	DISK 1	DISK 2	DISK 3	DISK 4
STRIPE 0	0	1	2	3	P0
STRIPE 1	4	5	6	7	P1
STRIPE 2	8	9	10	11	P2
STRIPE 3	12	13	14	15	P3
STRIPE 4	16	17	18	19	P4
STRIPE 5	20	24	28	32	P5
	21	25	29	33	P6
	22	26	30	34	P7
	23	27	31	35	P8
STRIPE 6	36	40	44	48	P9
	37	41	45	49	P10
	38	42	46	50	P11
	39	43	47	51	P12
STRIPE 7	52	56	60	64	P13
	53	57	61	65	P14
	54	58	62	66	P15
	55	59	63	67	P16
STRIPE 8	68	72	76	80	P17
	69	73	77	81	P18
	70	74	78	82	P19
	71	75	79	83	P20

FIG. 8

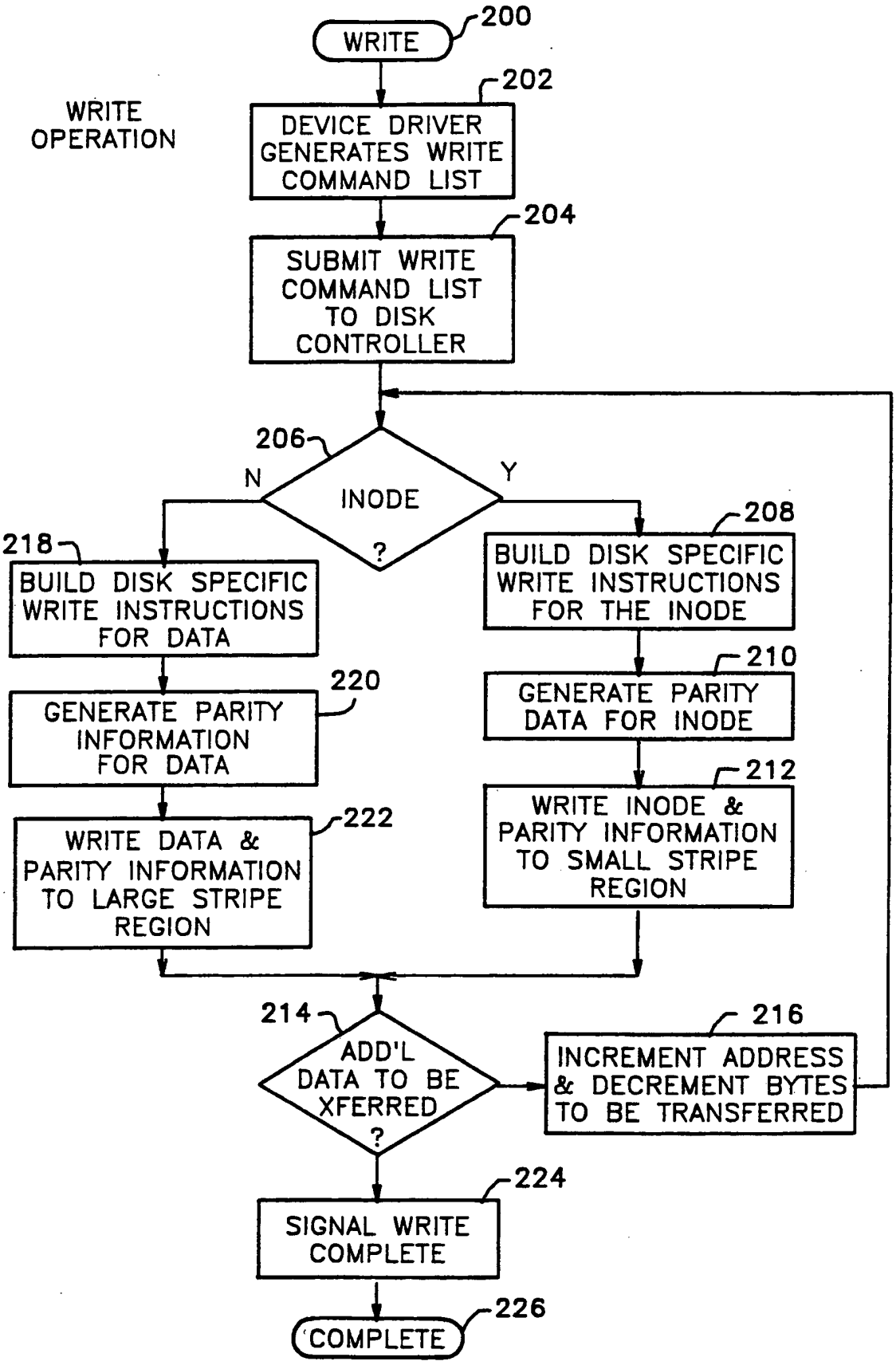
WRITE OPERATION
WRITE
200
DEVICE DRIVER GENERATES WRITE COMMAND LIST
202
SUBMIT WRITE COMMAND LIST TO DISK CONTROLLER
204
INODE ?
206
N
Y
BUILD DISK SPECIFIC WRITE INSTRUCTIONS FOR DATA
218
GENERATE PARITY INFORMATION FOR DATA
220
WRITE DATA & PARITY INFORMATION TO LARGE STRIPE REGION
222
BUILD DISK SPECIFIC WRITE INSTRUCTIONS FOR THE INODE
208
GENERATE PARITY DATA FOR INODE
210
WRITE INODE & PARITY INFORMATION TO SMALL STRIPE REGION
212
ADD'L DATA TO BE XFERRED ?
214
INCREMENT ADDRESS & DECREMENT BYTES TO BE TRANSFERRED
216
SIGNAL WRITE COMPLETE
224
COMPLETE
226

FIG. 9

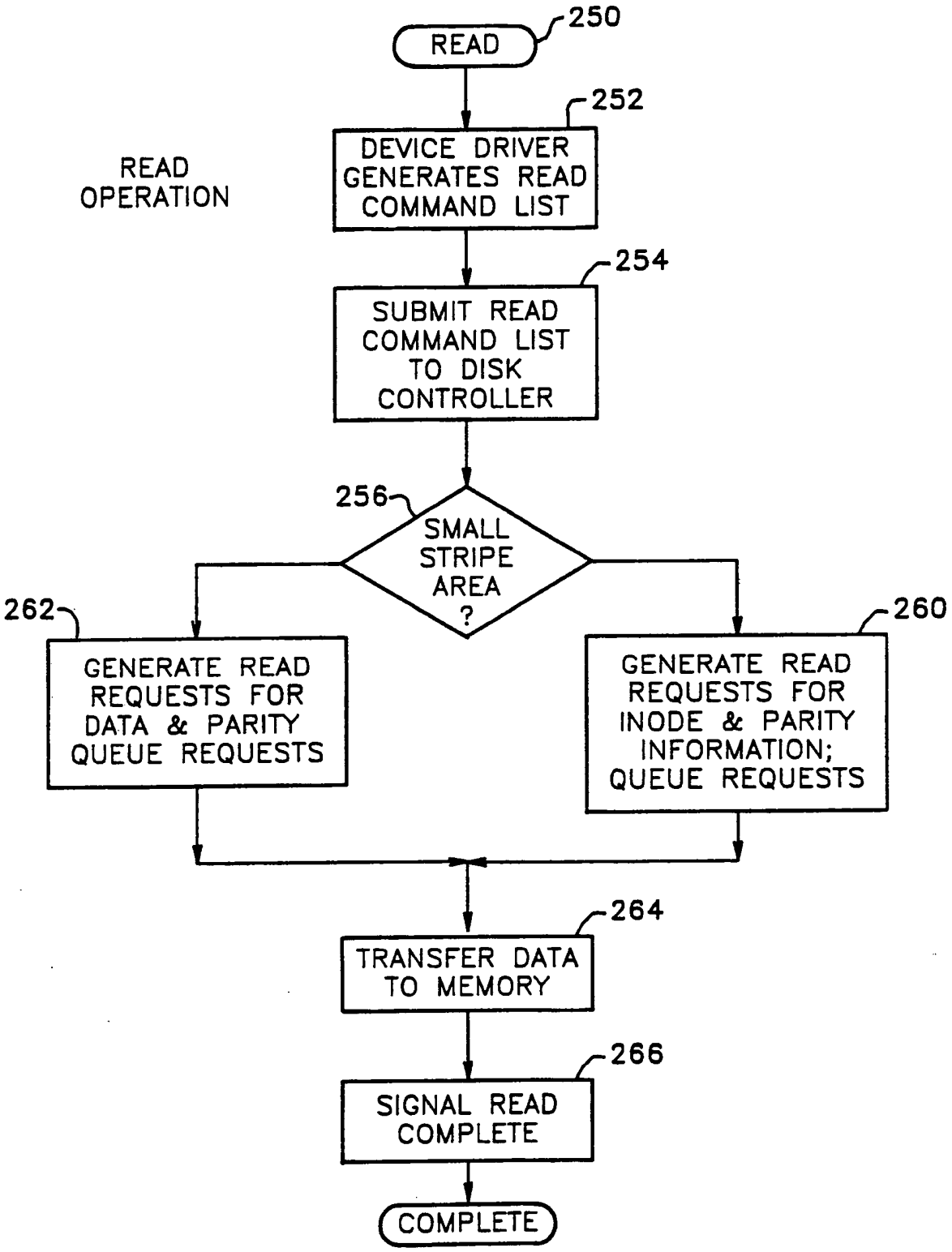
READ OPERATION
READ
250
DEVICE DRIVER GENERATES READ COMMAND LIST
252
SUBMIT READ COMMAND LIST TO DISK CONTROLLER
254
SMALL STRIPE AREA ?
256
GENERATE READ REQUESTS FOR DATA & PARITY QUEUE REQUESTS
262
GENERATE READ REQUESTS FOR INODE & PARITY INFORMATION; QUEUE REQUESTS
260
TRANSFER DATA TO MEMORY
264
SIGNAL READ COMPLETE
266
COMPLETE

FIG. 10

International Application No PCT/US 92/11203

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all)[6]

According to International Patent Classification (IPC) or to both National Classification and IPC

Int.Cl. 5 G06F3/06; G11B20/18; G06F11/10

II. FIELDS SEARCHED

Minimum Documentation Searched[7]

Classification System	Classification Symbols
Int.Cl. 5	G06F ; G11B

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched[8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT[9]

Category °	Citation of Document,[11] with indication, where appropriate, of the relevant passages[12]	Relevant to Claim No.[13]
A	EP,A,0 249 091 (IBM) 16 December 1987 see column 2, line 8 - line 30 see column 3, line 13 - line 27 see column 4, line 11 - column 5, line 8; figures 1,2	1,5,6,7, 9,11,12
A	PERFORMANCE EVALUATION REVIEW vol. 18, no. 1, May 1990, CA,US pages 74 - 85 CHEN ET AL 'AN EVALUATION OF REDUNDANT ARRAYS OF DISKS USING AN AMDAHL 5890' see page 77, left column, line 30 - right column, line 19	1,2,5, 7-10,12

° Special categories of cited documents :[10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search	Date of Mailing of this International Search Report
03 MAY 1993	25. 05. 93
International Searching Authority	**Signature of Authorized Officer**
EUROPEAN PATENT OFFICE	MOENS R.A.

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO.

US 9211283
SA 68644

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information. 03/05/93

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP-A-0249091	16-12-87	US-A- 4761785	02-08-88
		CA-A- 1270333	12-06-90
		JP-A- 62293355	19-12-87